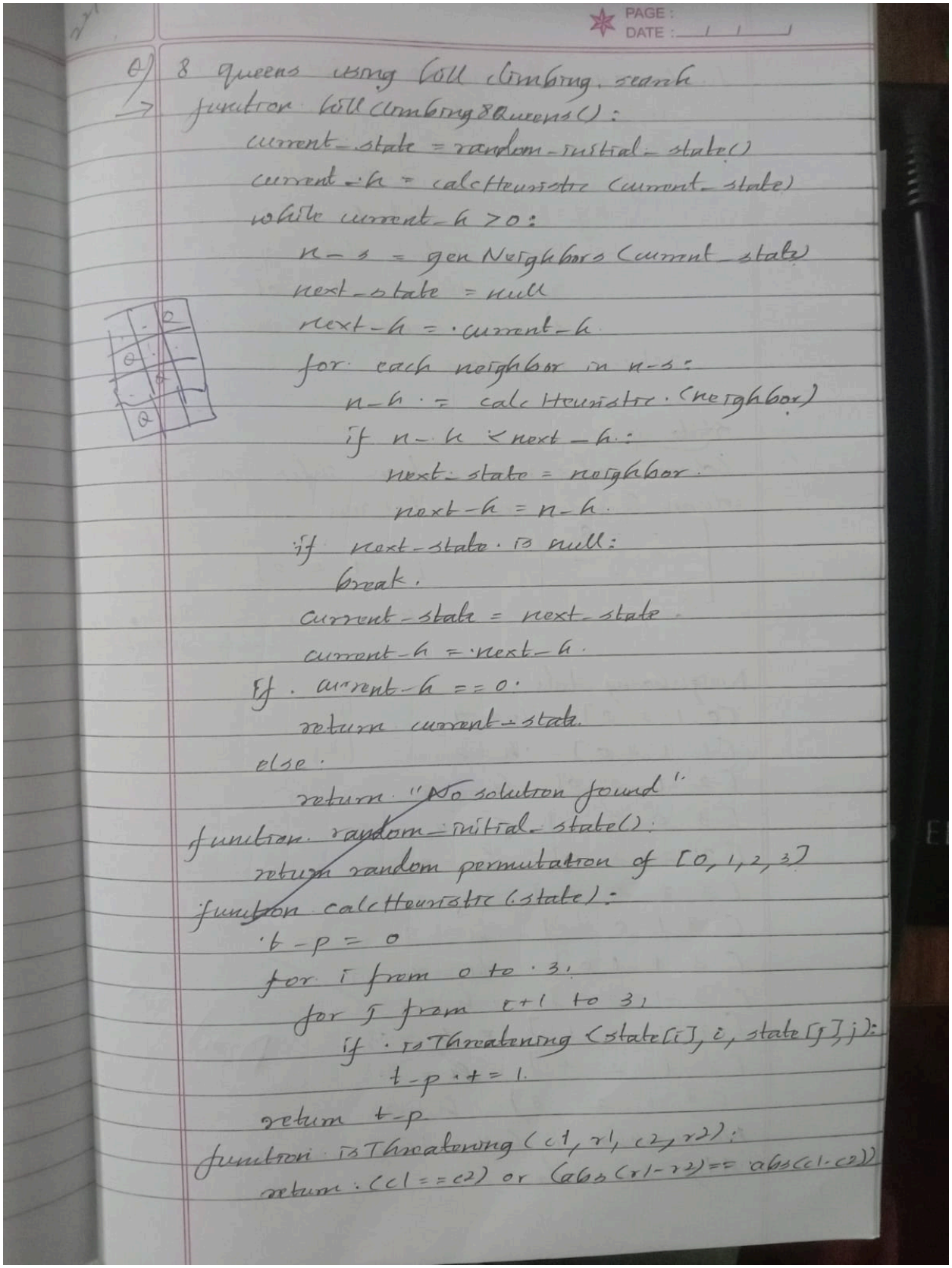
N Queens using Hill Climbing : Algorithm



Q) 8 Queens using hill climbing search

```
function hillClimbing8Queens():
    current_state = random_initial_state()
    current_h = calcHeuristic(current_state)
    while current_h > 0:
        n_s = genNeighbors(current_state)
        next_state = null
        next_h = current_h
        for each neighbor in n_s:
            n_h = calcHeuristic(neighbor)
            if n_h < next_h:
                next_state = neighbor
                next_h = n_h
        if next_state is null:
            break
        current_state = next_state
        current_h = next_h
    if current_h == 0:
        return current_state
    else:
        return "No solution found"
function random_initial_state():
    return random permutation of [0, 1, 2, 3]
function calcHeuristic(state):
    t_p = 0
    for i from 0 to 3:
        for j from i+1 to 3:
            if isThreatening(state[i], i, state[j], j):
                t_p += 1
    return t_p
function isThreatening(c1, r1, c2, r2):
    return (c1 == c2) or (abs(r1 - r2) == abs(c1 - c2))
```

PAGE :
DATE : / /

```
function generateNeighbors(state):
    neighbors = []
    for row from 0 to 3:
        for col from 0 to 3:
            if col != state[row]:
                new_state = state.copy()
                new_state[row] = col
                neighbors.append(new_state)
    return neighbors
```

State space

Column index representation, index represents column & value

State Space Diagram

neighbors.append(new_state)
return neighbors

State space
Column index representation, index represents column & value represents row.

$$\begin{bmatrix} - & - & - & Q \\ - & Q & - & - \\ - & - & Q & - \\ Q & - & - & - \end{bmatrix} \Rightarrow [3\ 1\ 2\ 0]$$

0 1 2 3

Heuristic h = 2 (Queens under attack)

Neighboring states.

[0 1 2 0] h = 4
[1 1 2 0] h = 2.
[2 1 2 0] h = 3
[3 0 2 0] h = 2.
[3 2 2 0] h = 4
[3 3 2 0] h = 3
[3 1 0 0] h = 3
[3 1 1 0] h = 4.
[3 1 3 0] h = 2.
[3 1 2 1] h = 3
[3 1 2 2] h = 2.
[3 1 2 3] h = 4.

No neighbor is better than initial.
Algorithm ends.

Code

```
import random

def calculate_heuristic(state):
    """Calculates the number of threatening pairs of queens."""
    threatening_pairs = 0
    size = len(state)
    for i in range(size):
        for j in range(i + 1, size):
            if is_threatening(state[i], i, state[j], j):
                threatening_pairs += 1
    return threatening_pairs

def is_threatening(row1, col1, row2, col2):
    """Checks if two queens threaten each other."""
    return (row1 == row2) or (abs(col1 - col2) == abs(row1 - row2))

def generate_neighbors(state):
    """Generates all possible neighbor states by moving queens to different rows in the same
column."""
    neighbors = []
    for col in range(len(state)):
        for row in range(len(state)):
            if row != state[col]:  # Don't move to the same row
                new_state = state.copy()
                new_state[col] = row  # Move queen to new row
                neighbors.append(new_state)
    return neighbors

def hill_climbing(size):
    """Main function to solve the n-Queens problem using hill climbing."""
    current_state = [3, 1, 2, 0]
    current_h = calculate_heuristic(current_state)
    print(current_h)
    while current_h > 0:  # While there are threatening pairs
        neighbors = generate_neighbors(current_state)
        next_state = None
        next_h = current_h

        for neighbor in neighbors:
            neighbor_h = calculate_heuristic(neighbor)
            print(neighbor)
            print(neighbor_h)
```

```
            # Check if this neighbor is better
            if neighbor_h < next_h:
                next_state = neighbor
                next_h = neighbor_h

        if next_state is None:
            print("No better neighbor exists")# No better neighbor found
            break

        current_state = next_state
        current_h = next_h

    if current_h == 0:
        return current_state  # Solution found
    else:
        return "No solution found."

# Example usage
size = 4
solution = hill_climbing(size)
print("Solution for 4-Queens:", solution)
```

Output :

```
2
[0, 1, 2, 0]
4
[1, 1, 2, 0]
2
[2, 1, 2, 0]
3
[3, 0, 2, 0]
2
[3, 2, 2, 0]
4
[3, 3, 2, 0]
3
[3, 1, 0, 0]
3
[3, 1, 1, 0]
4
[3, 1, 3, 0]
2
[3, 1, 2, 1]
3
[3, 1, 2, 2]
2
[3, 1, 2, 3]
4
No better neighbor exists
Solution for 4-Queens: No solution found.
```